Interface description

VISSIM 6.00-00
DriverModel DLL Interface

Karlsruhe, 2014-07-29

PTV AG

Document information

Title	Driver Model DLL Interface
Version:	VISSIM 6.00
Author:	Stefan Hengst
Created on:	2006-02-10
Edited by:	Lukas Kautzsch
Last saved:	2014-07-29

Contents

1 General

The External Driver Model DLL Interface of VISSIM provides the option to replace the internal driving behavior by a fully user-defined behavior for some or all vehicles in a simulation run. The user-defined algorithm must be implemented in a DLL written in C/C++ which contains specific functions (as specified below). During a simulation run, VISSIM calls the DLL code for each affected vehicle in each simulation time step to determine the behavior of the vehicle. VISSIM passes the current state of the vehicle and its surroundings to the DLL and the DLL computes the acceleration / deceleration of the vehicle and the lateral behavior (mainly for lane changes) and passes the updated state of the vehicle back to VISSIM.

The external driver model can be activated for each vehicle type separately in the dialog box “Vehicle Type” by checking the checkbox "Use external driver model” on the tab page “External Driver Model" and selecting a driver model DLL file and optionally a parameter file to be used. If this option is checked, the driving behavior of all vehicles of this vehicle type will be calculated by the selected DLL.
A subdirectory DriverModelData\ must exist in the directory of vissim.exe in order to avoid a warning message when VISSIM is started.

2 DLL Interface

For the implementation of the DLL, three source code files are provided:

- DriverModel.h: Header file for a driver model DLL.
  This file should not be changed. It contains the definitions of all "type" and "number" constants used by VISSIM in calls of the “DriverModel*” DLL functions which are declared here, too.
- DriverModel.cpp: Main source file of a driver model DLL.
  This file is the place where calculations or calls of functions of the driving behavior model algorithm should be added. Provided is a dummy version which does "nothing" (only passes the behavior suggested by VISSIM back to VISSIM). The preprocessor #define DRIVERMODEL_EXPORTS must be set in the compiler options for compilation of DriverModel.cpp! (This is included in the provided project file – see below.)
- DriverModel.vcxproj: Visual C++ 2010 project file for a driver model DLL. This file can be used if the DLL is to be created with Microsoft Visual C++.

A driver model DLL must contain and export 3 functions which are called from VISSIM: `DriverModelSetValue`, `DriverModelGetValue` and `DriverModelExecuteCommand`.
The signature of these functions and their meaning is as follows:

```
int  DriverModelSetValue (long   type,
                          long   index1,
                          long   index2,
                          long   long_value,
                          double double_value,
                          char   string_value);
```

VISSIM passes the current value of the data item indicated by `type` and (for most types) indexed by `index1` and sometimes `index2`. The value is passed in `long_value`, `double_value` or `string_value`, depending on `type`.
The code in the function must make sure to save the value somewhere if it is required later for the driving behavior calculation because the next call of this function from VISSIM will overwrite the local parameter.
(In the provided dummy DLL the values suggested by VISSIM (via several calls to `DriverModelSetValue`) are saved in global variables in order to be able to send them back when VISSIM calls `DriverModelGetValue` after the one call of `DriverModelExecuteCommand (DRIVER_COMMAND_MOVE_DRIVER)` per vehicle per time step.)
The function must return 1 for all types which are not marked as optional in the documentation below. For optional types, it can return 0 to inform VISSIM that it doesn't handle this type.

```
int  DriverModelGetValue (long   type,
                          long   index1,
                          long   index2,
                          long   *long_value,
                          double *double_value,
                          char   **string_value);
```

VISSIM retrieves the value of the data item indicated by `type` and (for most types) indexed by `index1` and sometimes `index2`. Before returning from this function, the value must be written to either `*long_value`, `*double_value` or `*string_value`, depending on the data type of the data item.
The function must return 1 for all types which are not marked as optional in the documentation below. For optional types, it can return 0 to inform VISSIM that it doesn't handle this type.

```
int  DriverModelExecuteCommand (long number);
```

VISSIM requests execution of the command indicated by `number`.
Currently available command constants are `DRIVER_COMMAND_INIT`, `DRIVER_COMMAND_CREATE_DRIVER`, `DRIVER_COMMAND_MOVE_DRIVER` and `DRIVER_COMMAND_KILL_DRIVER`. The function must return 1 for all these commands lest VISSIM stop the simulation run.

Before VISSIM requests execution of one of the available commands (`Init`, `CreateDriver`, `MoveDriver`, `KillDriver`) of the DLL there are always several calls of the DLL function `DriverModelSetValue`, one for each data item that might be used by the DLL when executing the command.

After the command `MoveDriver` has finished computation, the resulting state of the vehicle is fetched from the DLL in a similar manner again by several calls of `DriverModelGetValue`.

For a complete list of defined values for `type` and `number` see `DriverModel.h`.

3 Commands

There are 4 commands that a driver model DLL must implement: `Init`, `CreateDriver`, `MoveDriver` and `KillDriver`.

3.1 Init

This command is executed through a call of
`DriverModelExecuteCommand (DRIVER_COMMAND_INIT)`
at the start of a VISSIM simulation run to initialize the driver model DLL.
Several basic parameters are passed to the DLL before this through
`DriverModelSetValue ()` (shortened "`Set`" below),
and some values are retrieved from the DLL through
`DriverModelGetValue ()` ("`Get`").

The sequence of calls to the DLL is as follows:

Once per vehicle type which uses that DLL:

Get DRIVER_DATA_STATUS (optional)
Get DRIVER_DATA_STATUS_DETAILS (only if STATUS is not 0; optional)
Set DRIVER_DATA_PATH
Set DRIVER_DATA_PARAMETERFILE (optional)
Set DRIVER_DATA_TIMESTEP
Set DRIVER_DATA_TIME
Set DRIVER_DATA_VEH_TYPE
Get DRIVER_DATA_WANTS_SUGGESTION
Get DRIVER_DATA_SIMPLE_LANECHANGE

Then only once per DLL:

Execute DRIVER_COMMAND_INIT
Get DRIVER_DATA_STATUS (optional)
Get DRIVER_DATA_STATUS_DETAILS (only if STATUS is not 0; optional)

3.2 CreateDriver

`DriverModelExecuteCommand (DRIVER_COMMAND_CREATE_DRIVER)` is called from VISSIM during the simulation run whenever a new vehicle is set into the network in VISSIM (at the start of a time step, from a vehicle input, a PT line or a parking lot). In the same time step, a command `MoveDriver` for the same vehicle will follow later.

The sequence of calls to the DLL is as follows:

Set DRIVER_DATA_TIMESTEP
Set DRIVER_DATA_TIME
Set DRIVER_DATA_VEH_TYPE = VehicleTypeNumber
Set DRIVER_DATA_VEH_ID = NumberOfNewVehicle
Set DRIVER_DATA_VEH_DESIRED_VELOCITY = InitialDesiredVelocity
Set DRIVER_DATA_VEH_X_COORDINATE
Set DRIVER_DATA_VEH_Y_COORDINATE
Set DRIVER_DATA_VEH_REAR_X_COORDINATE
Set DRIVER_DATA_VEH_REAR_Y_COORDINATE

Execute DRIVER_COMMAND_CREATE_DRIVER

3.3 MoveDriver

`DriverModelExecuteCommand (DRIVER_COMMAND_MOVE_DRIVER)` is called from VISSIM during the simulation run once per stime step for each vehicle of a vehicle type which uses this DriverModel DLL. Before this call, there are many calls of `DriverModelSetValue ()` to pass the current state of the vehicle and its surroundings to the DLL. After the execution of the command, there are several calls of `DriverModelGetValue ()` to retrieve the new values for acceleration and lateral behavior from the DLL. Before any vehicle specific data is exchanged, VISSIM passes the states of all signal groups and priority rules to the DLL.

The sequence of calls to the DLL is as follows:

Global data

Set DRIVER_DATA_TIMESTEP
Set DRIVER_DATA_TIME

For each SC (passed in index1), for each signal head (passed in index2):
Set DRIVER_DATA_SIGNAL_STATE =
red = 1, amber = 2, green = 3, red/amber = 4, amber flashing = 5, off = 6, other = 0

For each priority rule section (“yield sign”) (index 1 = 0; number passed in index2):
Set DRIVER_DATA_SIGNAL_STATE =
red = 1, green = 3

Vehicle specific data

For each vehicle of a vehicle type using this driver model DLL:

Data of the subject vehicle

Set DRIVER_DATA_TIMESTEP =
time step length [simulation seconds] [0.1 .. 1.0]
Set DRIVER_DATA_TIME =
current simulation time (simulation seconds since simulation start)
Set DRIVER_DATA_VEH_ID =
ID of the vehicle to be moved
Set DRIVER_DATA_VEH_LANE =
current lane number (rightmost = 1)
Set DRIVER_DATA_VEH_ODOMETER =
total elapsed distance in the network [m]
Set DRIVER_DATA_VEH_LANE_ANGLE =
angle relative to the middle of the lane [rad]
Set DRIVER_DATA_VEH_LATERAL_POSITION =
distance of the front end from the middle of the lane [m]
Set DRIVER_DATA_VEH_VELOCITY =
current speed [m/s]
Set DRIVER_DATA_VEH_ACCELERATION =
current acceleration [m/s²]
Set DRIVER_DATA_VEH_LENGTH =
vehicle length [m]
Set DRIVER_DATA_VEH_WIDTH =
vehicle width [m]
Set DRIVER_DATA_VEH_WEIGHT =
vehicle weight [kg]
Set DRIVER_DATA_VEH_MAX_ACCELERATION =
maximum possible acceleration [m/s²]
Set DRIVER_DATA_VEH_TURNING_INDICATOR =
left = 1, right = -1, none = 0, both = 2
Set DRIVER_DATA_VEH_CATEGORY =
car = 1, truck = 2, bus = 3, tram = 4, pedestrian = 5, bike = 6
Set DRIVER_DATA_VEH_COLOR =
vehicle color (24 bit RGB value)
Set DRIVER_DATA_VEH_PREFERRED_REL_LANE =
positive = left, 0 = current lane, negative = right
Set DRIVER_DATA_VEH_USE_PREFERRED_LANE =
0 = only preferable (e.g. European highway with right-side rule),

1 = necessary (e.g. before a connector)
Set DRIVER_DATA_VEH_DESIRED_VELOCITY =
desired speed [m/s]
Set DRIVER_DATA_VEH_X_COORDINATE =
world coordinate X (vehicle front end)
Set DRIVER_DATA_VEH_Y_COORDINATE =
world coordinate Y (vehicle front end)
Set DRIVER_DATA_VEH_REAR_X_COORDINATE =
world coordinate X (vehicle rear end), optional
Set DRIVER_DATA_VEH_REAR_Y_COORDINATE =
world coordinate Y (vehicle rear end), optional
Set DRIVER_DATA_VEH_TYPE =
vehicle type number (user defined)
Set DRIVER_DATA_VEH_CURRENT_LINK =
current link number

Only if `DriverModelSetValue (`DRIVER_DATA_VEH_CURRENT_LINK`)` returned 1: For each link in the vehicle's route/path:
Set DRIVER_DATA_NEXT_LINKS =
link number

Set DRIVER_DATA_VEH_ACTIVE_LANE_CHANGE =
direction of an active lane change movement
(+1 = to the left, 0 = none, -1 = to the right), optional
Set DRIVER_DATA_VEH_REL_TARGET_LANE =
target lange (+1 = next one left, 0 = current lane, -1 = next one right), optional

Data of the nearby vehicles

For each nearby vehicle (up to two each downstream and upstream, on up to 2 lanes on both sides and the current lane) several values are passed from VISSIM:

For DRIVER_DATA_NVEH_* index1 and index2 are used as follows:
index1 = relative lane: +2 = second lane to the left, +1 = next lane to the left,
0 = current lane,
-1 = next lane to the right, -2 = second lane to the right
index2 = relative position: positive = downstream (+1 next, +2 second next)
negative = upstream (-1 next, -2 second next)

First, for each *possible* nearby vehicle position (each index combination) an initialization:

Set DRIVER_DATA_NVEH_ID = -1

Then, for each *existing* nearby vehicle the real data:

Set DRIVER_DATA_NVEH_ID =
vehicle number
Set DRIVER_DATA_NVEH_LANE_ANGLE =
angle relative to the middle of the lane [rad] (positive = turning left)
Set DRIVER_DATA_NVEH_LATERAL_POSITION =
distance of the front end from the middle of the lane [m]
(positive = left of the middle, negative = right)
Set DRIVER_DATA_NVEH_DISTANCE =
gross distance [m] (front end to front end)
Set DRIVER_DATA_NVEH_REL_VELOCITY =
speed difference [m/s] (veh. speed - nveh. speed)
Set DRIVER_DATA_NVEH_ACCELERATION =
current acceleration [m/s²]
Set DRIVER_DATA_NVEH_LENGTH =
vehicle length [m]
Set DRIVER_DATA_NVEH_WIDTH =
vehicle width [m]
Set DRIVER_DATA_NVEH_WEIGHT =
vehicle weight [kg]
Set DRIVER_DATA_NVEH_TURNING_INDICATOR =
left = 1, right = -1, none = 0, both = 2
Set DRIVER_DATA_NVEH_CATEGORY =
car = 1, truck = 2, bus = 3, tram = 4, pedestrian = 5, bike = 6
Set DRIVER_DATA_NVEH_LANE_CHANGE =
direction of a current lane change
(+1 = to the left, 0 = none, -1 = to the right)

Data of the current link

Set DRIVER_DATA_NO_OF_LANES =
number of lanes of the link the vehicle is currently on

Data of all lanes of the current link of the subject vehicle

For each lane of the current link of the vehicle, several values are passed from VISSIM:

For DRIVER_DATA_LANE_* index1 and index2 are used as follows:
index1 = lane number (rightmost = 1), index2 is irrelevant.

Set DRIVER_DATA_LANE_WIDTH =
lane width [m]
Set DRIVER_DATA_LANE_END_DISTANCE =
distance to end of lane [m] (can be emergency stop position before connector, negative = no end of lane in visibility range)

Data of the current and upcoming environment

Set DRIVER_DATA_RADIUS =
current curve radius [m]
Set DRIVER_DATA_MIN_RADIUS =
minimum curve radius [m] in visibility range
Set DRIVER_DATA_DIST_TO_MIN_RADIUS =
distance [m] to spot of minimum curve radius
Set DRIVER_DATA_SLOPE =
current slope (negative = drop)
Set DRIVER_DATA_SLOPE_AHEAD =
slope at end of visibility range

Data of the next signal head

Set DRIVER_DATA_SIGNAL_DISTANCE =
distance [m] to next signal head (negative = no signal head visible)
(Before VISSIM 5.30, it's undefined if signal data is sent at all if no signal heads are inside the visibility distance.)
Set DRIVER_DATA_SIGNAL_STATE =
red = 1, amber = 2, green = 3, red/amber = 4, amber flashing = 5, off = 6, green arrow = 7
Set DRIVER_DATA_SIGNAL_STATE_START =
simulation time [s] when signal changed to current state

Data of the next reduced speed area or previous desired speed decision

Set DRIVER_DATA_SPEED_LIMIT_DISTANCE =
distance [m] to "speed limit sign" (reduced speed area: real distance, desired speed decision: 1.0 m when just passed, negative: no sign visible)
(Before VISSIM 5.30, it's undefined if speed limit data is sent at all if no reduced speed area is inside the visibility distance and no desired speed decision has been passed in the previous time step.)
Set DRIVER_DATA_SPEED_LIMIT_VALUE =
speed limit [km/h] (0 = end of reduced speed area)

Behavior data suggested by VISSIM's internal model

These values are passed only if `*long_value` was set to 1 in the call of `DriverModelGetValue (DRIVER_DATA_WANTS_SUGGESTION)`.

Set DRIVER_DATA_DESIRED_ACCELERATION =
desired acceleration [m/s²] in next time step
Set DRIVER_DATA_DESIRED_LANE_ANGLE =
desired angle relative to the middle of the lane [rad] (positive = turning left)
Set DRIVER_DATA_ACTIVE_LANE_CHANGE =
direction of an active lane change movement (+1 = to the left, 0 = none,

-1 = to the right, must be != 0 while lane change is not completed,
will be used for NVEH_LANE_CHANGE seen from other vehicles)
Set DRIVER_DATA_REL_TARGET_LANE =
target lane (+1 = next one left, 0 = current lane, -1 = next one right)

Execute Move Command

Execute DRIVER_COMMAND_MOVE_DRIVER

Pass new data calculated by the behavior model in the DLL back to VISSIM

Get DRIVER_DATA_VEH_TURNING_INDICATOR =
left = 1, right = -1, none = 0, both = 2
Get DRIVER_DATA_VEH_DESIRED_VELOCITY =
desired speed [m/s]
Get DRIVER_DATA_VEH_COLOR =
vehicle color (24 bit RGB value)
Get DRIVER_DATA_DESIRED_ACCELERATION =
new acceleration [m/s²], optional
[This is limited by VISSIM to the minimum of desired acceleration and maximum acceleration and to the maximum deceleration for the vehicle at the current speed. If this is set, VISSIM shows the interaction state of the vehicle as “PELOPS” in the vehicle record.]
Get DRIVER_DATA_DESIRED_LANE_ANGLE =
desired angle relative to the middle of the lane [rad] (positive = turning left), optional
[If `*long_value` was set to 1 in the call of `DriverModelGetValue (DRIVER_DATA_SIMPLE_LANECHANGE)` this angle does not need to be calculated by the DLL and the return value of `DriverModelGetValue(DRIVER_DATA_DESIRED_LANE_ANGLE)` should be zero.]
Get DRIVER_DATA_ACTIVE_LANE_CHANGE =
direction of an active lane change movement (+1 = to the left, 0 = none,
-1 = to the right, must be != 0 while lane change is not completed)
[If `*long_value` was set to 1 in the call of `DriverModelGetValue (DRIVER_DATA_SIMPLE_LANECHANGE)` setting this to +1 or -1 is sufficient to start a lane change which will be completed automatically by VISSIM.]
Get DRIVER_DATA_REL_TARGET_LANE =
target lane (+1 = next one left, 0 = current lane, -1 = next one right)
[This is used by VISSIM only if `*long_value` was set to 0 in the call of `DriverModelGetValue (DRIVER_DATA_SIMPLE_LANECHANGE)`.]

3.4 KillDriver

`DriverModelExecuteCommand (DRIVER_COMMAND_KILL_DRIVER)` is called from VISSIM when a vehicle reaches its destination and thus leaves the network (so memory which the DLL might have allocated for that vehicle can be freed):

Set DRIVER_DATA_VEH_ID =
    ID of the vehicle to be killed

Execute DRIVER_COMMAND_KILL_DRIVER

4 Lane Change

4.1 General

The driver model DLL interface provides 2 different ways to control lane changes of vehicles affected by the external dll:

- Simple lane change
- Full control over the lane change

If simple lane change is selected, the driver model DLL only needs to initiate a lane change for a vehicle. VISSIM assumes control of the lateral behavior of this vehicle while the lane change proceeds and informs the driver model DLL when it is done.

Without simple lane change the driver model DLL has full control of the vehicle and manages the lane change on its own – the driver model DLL must inform VISSIM about the current state of the vehicle.

How lane changes are actually handled is determined by the data types DRIVER_DATA_SIMPLE_LANECHANGE and DRIVER_DATA_WANTS_SUGGESTION.

The lane change itself involves the data types DRIVER_DATA_VEH_ACTIVE_LANE_CHANGE, DRIVER_DATA_VEH_REL_TARGET_LANE, DRIVER_DATA_ACTIVE_LANE_CHANGE, DRIVER_DATA_REL_TARGET_LANE and DRIVER_DATA_DESIRED_LANE_ANGLE.

4.2 Simple lane change - handled by VISSIM

This mode is chosen by setting `*long_value` to 1 as result of VISSIM's initial request for DRIVER_DATA_SIMPLE_LANECHANGE.

If the driver model sets `*long_value` to 1 as result for DRIVER_DATA_WANTS_SUGGESTION as well, VISSIM will send a suggestion whenever it detects that a lane change is necessary. As long as the driver model passes back those suggestions VISSIM has complete control of lane changes.

A lane change can be executed by the driver model DLL the following way:

1. The driver model sets DRIVER_DATA_ACTIVE_LANE_CHANGE to 1 (to the left) or -1 (to the right).
2. VISSIM starts the lane changing mode for the current vehicle. While in this mode, the vehicle ignores the values sent from the DLL for DRIVER_DATA_ACTIVE_LANE_CHANGE, DRIVER_DATA_REL_TARGET_LANE and DRIVER_DATA_DESIRED_LANE_ANGLE.
3. VISSIM sets DRIVER_DATA_VEH_REL_TARGET_LANE, DRIVER_DATA_VEH_LANE and DRIVER_DATA_VEH_LANE_ANGLE while in lane changing mode. (DRIVER_DATA_VEH_REL_TARGET_LANE is set to zero when the middle of the front end of the vehicle has reached the target lane, and DRIVER_DATA_VEH_LANE is set to the new lane at this time, too.)
4. When VISSIM has finished the lane change (i.e. when the whole width of the vehicle is on the new lane), it sends 0 as value of DRIVER_DATA_VEH_ACTIVE_LANE_CHANGE. (This does not necessarily mean that the vehicle has reached its desired lateral position within the destination lane.)

Hints:

- A current lane change cannot be interrupted when in simple lane change mode.
- Do not send back VISSIM's suggestions for DRIVER_DATA_DESIRED_LANE_ANGLE when no lane change is currently active, instead, return zero from `DriverModelGetValue()`.

4.3 Lane change - handled by the driver model DLL

This mode is chosen by setting `*long_value` to 0 as result of VISSIM's initial request for DRIVER_DATA_SIMPLE_LANECHANGE.

A lane change can be executed by the driver model DLL the following way:

1. The driver model DLL sets DRIVER_DATA_ACTIVE_LANE_CHANGE, DRIVER_DATA_REL_TARGET_LANE and DRIVER_DATA_DESIRED_LANE_ANGLE to the desired values.
2. VISSIM moves the vehicle according to these values and sets DRIVER_DATA_VEH_REL_TARGET_LANE to zero when the middle of the front end of the vehicle has reached the target lane and sets DRIVER_DATA_VEH_LANE to the new lane number at this time, too. (If there is no new lane on this side, VISSIM sets the vehicle back to the middle of the old lane and stops the lane change itself, setting DRIVER_DATA_ACTIVE_LANE_CHANGE and DRIVER_DATA_DESIRED_LANE_ANGLE to zero.)
3. The driver model DLL has to determine itself when the lane change is over – it has to reset the values for DRIVER_DATA_ACTIVE_LANE_CHANGE and DRIVER_DATA_DESIRED_LANE_ANGLE to zero.

Hints:

- The driver model has full control in this mode – it can even interrupt a current lane change.
- VISSIM does not finish a lane change on its own in this mode. This means that the vehicle will continue moving laterally beyond the next lane if the driver model does not stop the lane change.